

# On the $k$ -Independence Required by Linear Probing and Minwise Independence\*

Mihai Pătraşcu<sup>†</sup>

Mikkel Thorup

AT&T Labs

## Abstract

We show that linear probing requires 5-independent hash functions for expected constant-time performance, matching an upper bound of [Pagh et al. STOC'07]. More precisely, we construct a 4-independent hash functions yielding expected logarithmic search time. For  $(1 + \varepsilon)$ -approximate minwise independence, we show that  $\Omega(\lg \frac{1}{\varepsilon})$ -independent hash functions are required, matching an upper bound of [Indyk, SODA'99]. We also show that the very fast 2-independent multiply-shift scheme of Dietzfelbinger [STACS'96] fails badly in both applications.

## 1 Introduction

The concept of  $k$ -wise independence was introduced by Wegman and Carter [23] in FOCS'79 and has been the cornerstone of our understanding of hash functions ever since. Formally, a family  $\mathcal{H} = \{h : [u] \rightarrow [t]\}$  of hash functions is  $k$ -independent if (1) for any distinct keys  $x_1, \dots, x_k \in [u]$ , the hash codes  $h(x_1), \dots, h(x_k)$  are independent random variables; and (2) for any fixed  $x$ ,  $h(x)$  is uniformly distributed in  $[t]$ .

As the concept of independence is fundamental to probabilistic analysis,  $k$ -independent functions are both natural and powerful in algorithm analysis. They allow us to replace the heuristic assumption of truly random hash functions with real (implementable) hash functions that are still “independent enough” to yield provable performance guarantees. We are then left with the natural goal of understanding the independence required by algorithms.

When first we have proved that  $k$ -independence suffices for a hashing-based randomized algorithm, then we are free to use *any*  $k$ -independent hash function. The canonical construction of a  $k$ -independent family is based on polynomials of degree  $k - 1$ . Let  $p \geq u$  be prime. Picking random  $a_0, \dots, a_{k-1} \in \{0, \dots, p - 1\}$ , the hash function is defined by:

$$h(x) = \left( (a_{k-1}x^{k-1} + \dots + a_1x + a_0) \bmod p \right) \bmod t$$

For  $p \gg t$ , the hash function is statistically close to  $k$ -independent.

Sometimes 2-independence suffices. For instance, if one implements a hash table by chaining, the time it takes to query  $x$  is proportional to the number of keys  $y$  colliding with  $x$  (i.e.  $h(x) = h(y)$ ).

---

\*A preliminary version of this paper was presented at *The 37th International Colloquium on Automata, Languages and Programming (ICALP'10)* [15].

<sup>†</sup>Passed away June 5, 2012

Independence	2	3	4	$\geq 5$
Query time	$\Theta(\sqrt{n})$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Construction time	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n)$	$\Theta(n)$

Table 1: Expected time bounds for linear probing with a bad family of  $k$ -independent hash functions. Construction time refers to the total time to insert  $n$  keys starting from an empty table.

Thus, pairwise independence of  $h(x)$  and  $h(y)$  is all we need for expected constant query time. We note that 2-independence also suffices for the 2-level hashing of Fredman et al. [8], yielding static hash tables with constant query time.

At the other end of the spectrum,  $O(\lg n)$ -independence suffices in a vast majority of applications. One reason for this is the Chernoff bounds of [18] for  $k$ -independent events, whose probability bounds differ from the full-independence Chernoff bound by  $2^{-\Omega(k)}$ . Another reason is that random graphs with  $O(\lg n)$ -independent edges [1] share many of the properties of truly random graphs.

In this paper, we study two compelling applications in which independence bigger than 2 is currently needed: linear probing and minwise-independent hashing. (The reader unfamiliar with these applications will find more details below.) For linear probing, Pagh et al. [13] showed that 5-independence suffices, thus giving the first realistic implementation of linear probing with formal guarantees. For minwise-independence, Indyk [10] showed that  $\varepsilon$ -approximation can be obtained using  $O(\lg \frac{1}{\varepsilon})$ -independence.

In both cases, it was known that 2-independence does not suffice [13, 3], and, indeed, the simplest family  $x \mapsto (ax + b) \bmod p$  provides a counterexample. However, a significant gap remained to the upper bounds.

In this paper, we close this gap, showing that both upper bounds are, in fact, tight. We do this by exhibiting carefully constructed families for which these algorithms fail: for linear probing, we give a 4-independent family that leads to  $\Omega(\lg n)$  expected query time; and for minwise independence, we give an  $\Omega(\lg \frac{1}{\varepsilon})$ -independent family that leads to  $2\varepsilon$  approximation. In fact, we will present a complete understanding of linear probing with low independence as summarized in Table 1.

**Concrete schemes.** Our results give a powerful understanding of a natural combinatorial resource (independence) for two important algorithmic questions. In other words, they are limits on how far the *paradigm* of independence can bring us. Note, however, that independence is only one property that concrete hash schemes have. In a particular application, a hash scheme can behave much better than its independence guarantees, if it has some other probabilistic property unrelated to independence. Obviously, proving that a concrete hashing scheme works is not as attractive as proving that every  $k$ -independent scheme works, including more efficient  $k$ -independent schemes found in the future. However, if low independence does not work, then a concrete scheme may be the best we can hope for.

The most practical 2-independent hash function is not the standard  $x \mapsto ((ax+b) \bmod p) \bmod t$ , but Dietzfelbinger’s multiply-shift scheme [6], which on some computers is 10 times as fast [20]. To hash  $w$ -bit integers to  $\ell$ -bit integers,  $\ell \leq w$ , the scheme picks two random  $2w$ -bit integers  $a$  and  $b$ , and computes  $(ax + b) \gg (2w - \ell)$ , where  $\gg$  denotes unsigned shift.

In this paper, we prove that linear probing with multiply-shift hashing suffers from  $\Omega(\lg n)$  expected running times on some input. Similarly, we show that minwise independent hashing may have a very large approximation error of  $\varepsilon = \Omega(\lg n)$ . While these results are not surprising, given

the “moral similarity” of multiply-shift and  $ax + b \bmod p$  schemes, they do require rather involved arguments. We feel this effort is justified, as it brings the theoretical lower bounds in line with programming reality.

**Later work.** In our continued search for efficient hashing schemes with good theoretical properties, we later considered simple tabulation hashing [16], which breaks fundamentally from polynomial hashing schemes. Tabulation based hashing is comparable in speed to multiply-shift hashing [6], but it uses much more space (polynomial instead of constant). It is only 3-independent, yet it does give constant expected time for linear probing and  $o(1)$ -approximate minwise hashing. It is the negative findings of the current paper that motivated us to look into the much more space consuming tabulation hashing.

The problems discovered here for minwise hashing with low independence also lead the second author to consider alternatives more suitable for low independence [21].

## 1.1 Technical Discussion: Linear Probing

Linear probing uses a hash function to map a set of keys into an array of size  $t$ . When inserting  $x$ , if the desired location  $h(x)$  is already occupied, the algorithm scans  $h(x) + 1, h(x) + 2, \dots$  until an empty location is found, and places  $x$  there. The query algorithm starts at  $h(x)$  and scans either until it finds  $x$ , or runs into an empty position, which certifies that  $x$  is not in the hash table. We assume constant load of the hash table, e.g. the number of keys is  $n \leq \frac{2}{3}t$ .

This classic data structure is one of the most popular implementation of hash tables, due to its unmatched simplicity and efficiency. The practical use of linear probing dates back at least to 1954 to an assembly program by Samuel, Amdahl, Boehme (c.f. [12]). On modern architectures, access to memory is done in cache lines (of much more than a word), so inspecting a few consecutive values typically translates into just one memory probe. Even if the scan straddles a cache line, the behavior will still be better than a second random memory access on architectures with prefetching. Empirical evaluations [2, 9, 14] confirm the practical advantage of linear probing over other known schemes, while cautioning [9, 22] that it behaves quite unreliably with weak hash functions (such as 2-independent). Taken together, these findings form a strong motivation for theoretical analysis.

Linear probing was first shown to take expected constant time per operation in 1963 by Knuth [11], in a report now considered the birth of algorithm analysis. However, this required truly random hash functions.

A central open question of Wegman and Carter [23] was how linear probing behaves with  $k$ -independence. Siegel and Schmidt [17, 19] showed that  $O(\lg n)$ -independence suffices. Recently, Pagh et al. [13] showed that even 5-independent hashing works. We now close this line of work, showing that 4-independence is not enough.

**Review of the 5-independence upper bound.** To better situate our lower bounds, we begin by reviewing the upper bound of [13]. Our proof here is much simpler than the one in [13] but assumes a load factor below  $2/3$ . A more elaborate proof considering all load factors is presented in [16].

The main probabilistic tool featuring in this analysis is a 4<sup>th</sup> moment bound. Consider throwing  $n$  balls into  $t$  bins uniformly. Let  $X_i$  be the probability that ball  $i$  lands in the first bin, and  $X = \sum_{i=1}^n X_i$  the number of balls in the first bin. We have  $\mu = \mathbf{E}[X] = \frac{n}{t}$ . Then, the  $k^{\text{th}}$  moment of  $X$  is defined as  $\mathbf{E}[(X - \mu)^k]$ .

As long as our placement of the balls is  $k$ -independent, the  $k^{\text{th}}$  moment is identical to the case of full independence. For instance, the  $4^{\text{th}}$  moment is:

$$\mathbf{E}[(X - \mu)^4] = \mathbf{E}\left[\left(\sum_i (X_i - \tfrac{1}{t})\right)^4\right] = \sum_{i,j,k,l} \mathbf{E}\left[(X_i - \tfrac{1}{t})(X_j - \tfrac{1}{t})(X_k - \tfrac{1}{t})(X_l - \tfrac{1}{t})\right].$$

The only question in calculating this quantity is the independence of sets of at most 4 items. Thus, 4-independence preserves the  $4^{\text{th}}$  moment of full randomness. Since  $\mathbf{E}[(X_i - \frac{1}{t})] = 0$ ,

$$\mathbf{E}[(X - \mu)^4] = \sum_i \mathbf{E}[(X_i - \tfrac{1}{t})^4] + \binom{4}{2} \sum_{i,j} \mathbf{E}[(X_i - \tfrac{1}{t})^2 (X_j - \tfrac{1}{t})^2].$$

Moments are a standard approach for bounding the probability of large deviations. Let's say that we expect  $\mu$  items in the bin, but have capacity  $2\mu$ ; what is the probability of overflow? A direct calculation shows that the  $4^{\text{th}}$  moment is  $\mathbf{E}[(X - \mu)^4] = O(\mu^2)$ . Then, by a Markov bound, the probability of overflow is  $\Pr[X \geq 2\mu] = \Pr[(X - \mu)^4 \geq \mu^4] = O(1/\mu^2)$ . By contrast, if we only have 2-independence, we can use the  $2^{\text{nd}}$  moment  $\mathbf{E}[(X - \mu)^2] = O(\mu)$  and obtain  $\Pr[X \geq 2\mu] = O(1/\mu)$ . Observe that the  $3^{\text{rd}}$  moment is not useful for this approach, since  $(X - \mu)^3$  can be negative, so Markov does not apply.

To apply moments to linear probing, we consider a perfect binary tree spanning the array  $[t]$  where  $t$  is a power of two. For notational convenience, we assume that the load factor is  $1/3$ , that is,  $n = t/3$ . A node at height  $h \leq \log_2 t$  has an interval of  $2^h$  array positions below it, and is identified with this interval. We expect at most  $2^h/3$  keys to be hashed to the interval (but more or less keys may end in the interval, since items are not always placed at their hash position). Call the node “near-full” if at least  $\frac{2}{3}2^h$  keys hash to its interval.

We will now bound the total time it takes to construct the hash table (the cost of inserting  $n$  distinct items). A run is an maximal interval of filled locations. If the table consists of runs of  $k_1, k_2, \dots$  keys ( $\sum k_i = n$ ), the cost of constructing it is bounded from above by  $O(k_1^2 + k_2^2 + \dots)$ . To bound these runs, we make the following crucial observation: if a run contains between  $2^h$  and  $2^{h+1}$  keys, then some node at height  $h-2$  above it is near-full. In fact, there will be such a near-full height  $h-2$  node whose last position is in the run.

For a proof, we study a run of length at least  $2^h$ . The run is preceded by an empty position, so all keys in the run are hashed to the run (but may appear later in the run than the position they hashed to). There are at least 4 consecutive height  $h-2$  nodes with their last position in the interval. Assume for a contradiction that none of these are near-full. The first node (whose first positions may not be in the run) contributes less than  $\frac{2}{3}2^{h-2}$  keys to the run (in the most extreme case, this many keys hash to the last position of that node). The subsequent nodes have all  $2^{h-2}$  positions in the run, but with less than  $\frac{2}{3}2^{h-2}$  keys hashing to these positions. Even with the maximal excess from the first node, we cannot fill the intervals of two subsequent nodes, so the run must stop before the end of the third node, contradicting that its last position was in the run.

Each node has its last position in at most one run, so the observation gives an upper bound on the cost: add  $O(2^{2h})$  for each near-full node at some height  $h$ . Denoting by  $p(h)$  the probability that a node on height  $h$  is near-full, the expected total cost over all heights is thus bounded by  $\sum_{h=0}^{\log_2 t} (t/2^h) \cdot p(h) \cdot 2^{2h} = O(n \cdot \sum_{h=0}^{\log_2 t} 2^h \cdot p(h))$ . Using the  $2^{\text{nd}}$  moment to bound  $p(h)$ , we obtain  $p(h) = O(2^{-h})$ , so the total expected cost with 2-independence is  $O(n \lg n)$ . However, the  $4^{\text{th}}$  moment gives  $p(h) = O(2^{-2h})$ , so the total expected cost with 4-independence is  $O(n)$ .

To bound the running time of one particular operation (query or insert  $q$ ), we want  $q$  to be independent of the construction. Thus, if the hash function is  $k$ -independent, then we view the hash of  $q$  as independent of a  $(k - 1)$ -independent construction. The expected time is then the average distance from a position to the end of the run containing it. There is also a constant to be added for empty positions, but we ignore that below. The average cost is then a fraction  $1/t$  ( $\leq 1/n$ ) of the sum of the squared run lengths, which is exactly what we bounded above. Thus, with 3-independent hashing, the expected operation time is  $O(\log n)$  while with 5-independent hashing, the expected operation time is  $O(1)$ . For load factors below  $2/3$ , this establishes the upper bounds from Table 1 except for the  $O(\sqrt{n})$  search time with 2-independent hashing.

**Our results.** Two intriguing questions pop out of this analysis. First, is the independence of the query really crucial? Perhaps one could argue that the query behaves like an average operation, even if it is not completely independent of everything else. Secondly, one has to wonder whether 3-independence suffices (by using something other than 3<sup>rd</sup> moment): all that is needed is a bound slightly stronger than 2<sup>nd</sup> moment in order to make the costs with increasing heights decay geometrically!

We answer both questions in strong negative terms. The complete understanding of linear probing with low independence is summarized in Table 1. Addressing the first question, we show that 4-independence cannot give expected time per operation better than  $\Omega(\lg n)$ , even though  $n$  operations take  $O(n)$  time. Our proof demonstrates an important phenomenon: even though most bins have low load, a particular key’s hash code could be correlated with the (uniformly random) choice of *which* bins have high load.

An even more striking illustration of this fact happens for 2-independence: the query time blows up to  $\Omega(\sqrt{n})$  in expectation, since we are left with no independence at all after conditioning on the query’s hash. This demonstrates a very large separation between linear probing and collision chaining, which enjoys  $O(1)$  query times even for 2-independent hash functions.

Addressing the second question, we show that 3-independence is not enough to guarantee even a construction time of  $O(n)$ . Thus, in some sense, the 4<sup>th</sup> moment analysis is the best one can hope for.

## 1.2 Technical Discussion: Minwise Independence

This concept was introduced by two classic algorithms: detecting near-duplicate documents [3, 4] and approximating the size of the transitive closure [5]. The basic step in these algorithms is estimating the size of the intersection of pairs of sets, relative to their union: for  $A$  and  $B$ , we want to find  $\frac{|A \cap B|}{|A \cup B|}$  (the *Jaccard similarity coefficient*). To do this efficiently, one can choose a hash function  $h$  and maintain  $\min h(A)$  as the sketch of an entire set  $A$ . If the hash function is truly random, we have  $\Pr[\min h(A) = \min h(B)] = \frac{|A \cap B|}{|A \cup B|}$ . Thus, by repeating with several hash functions, or by keeping the bottom  $k$  keys with one hash function, the Jaccard coefficient can be estimated up to a small approximation.

To make this idea work, the property that is required of the hash function is *minwise independence*. Formally, a family of functions  $\mathcal{H} = \{h : [u] \rightarrow [u]\}$  is said to be minwise independent if, for any set  $S \subset [u]$  and any  $x \notin S$ , we have  $\Pr_{h \in \mathcal{H}}[h(x) < \min h(S)] = \frac{1}{|S|+1}$ . In other words,  $x$  is the minimum of  $S \cup \{x\}$  only with its “fair” probability  $\frac{1}{|S|+1}$ .

As good implementations of exact minwise independent functions are not known, the definition is

relaxed to  $\varepsilon$ -minwise independent, where  $\Pr_{h \in \mathcal{H}}[h(x) < \min h(S)] = \frac{1 \pm \varepsilon}{|S|+1}$ . Using such a function, we will have  $\Pr[\min h(A) = \min h(B)] = (1 \pm \varepsilon) \frac{|A \cap B|}{|A \cup B|}$ . Thus, the  $\varepsilon$  parameter of the minwise family dictates the best approximation achievable in the algorithms (which *cannot* be improved by repetition).

Indyk [10] gave the only implementation of minwise independence with provable guarantees, showing that  $O(\lg \frac{1}{\varepsilon})$ -independent functions are  $\varepsilon$ -minwise independent.

His proof uses another tool enabled by  $k$ -independence: the inclusion-exclusion principle. Say we want to bound the probability that at least one of  $n$  events is “good.” We can define  $p(k) = \sum_{S \subseteq [n], |S|=k} \Pr[\text{all } S \text{ are good}]$ . Then, the probability that at least one event is good is, by inclusion-exclusion,  $p(1) - p(2) + p(3) - p(4) + \dots$ . If we only have  $k$ -independence ( $k$  odd), we can upper bound the series by  $p(1) - p(2) + \dots + O(p(k))$ . In the common scenario that  $p(k)$  decays exponentially with  $k$ , the trimmed series will only differ from the full independence case by  $2^{-\Omega(k)}$ . Thus,  $k$ -independence achieves bounds exponentially close to full independence, whenever probabilities can be computed by inclusion-exclusion. This turns out to be the case for minwise independence: we can express the probability that at least some key in  $S$  is below  $x$  by inclusion-exclusion.

In this paper, we show that, for any  $\varepsilon > 0$ , there exist  $\Omega(\lg \frac{1}{\varepsilon})$ -independent hash functions that are no better than  $\varepsilon$ -minwise independent. Indyk’s [10] simple analysis via inclusion-exclusion is therefore tight:  $\varepsilon$ -minwise independence requires  $\Omega(\lg \frac{1}{\varepsilon})$  independence.

## 2 Linear probing with $k$ -independence

We will now present our analysis of linear probing with different degrees of independence. We will present negative results complementing the positive findings from [13], reviewed above. When lower bounding query times, for simplicity we assume that the query is not among the stored keys. The cost of such an unsuccessful search is the distance to the end of the run the query hashes to.

### 2.1 Expected query time $\Theta(\sqrt{n})$ with 2-Independence

Above we saw that the expected construction time with 2-independence is  $O(n \log n)$ , so the average cost per key is  $O(\log n)$ . We will now define a 2-independent hash family such that the expected query time for some concrete key is  $\Theta(\sqrt{n})$ . The main idea of the proof is that the query can play a special role: even if most portions of the hash table are lightly loaded, the query can be correlated with the portions that *are* loaded. We assume that  $t$  is an odd power of two, and we store  $n = t/2$  keys. Then  $\sqrt{n}$  is also a power of two.

We think of the stored keys and the query key as given, and we want to find bad ways of distributing them 2-independently into the range  $[t]$ . To extend the hash function to the entire universe, all other keys are hashed totally randomly. We consider unsuccessful searches, i.e. the search key  $q$  is not stored in the hash table. The query time for  $q$  is the number of cells considered from  $h(q)$  up to the first empty cell. If, for some  $d$ , the interval  $Q = (h(q) - d, h(q)]$  has  $2d$  keys, then the search time is  $\Omega(d)$ .

Let  $d = 2\sqrt{n}$ ; this is a power of two dividing  $t$ . In our construction, we first pick the hash  $h(q)$  uniformly. We then divide the range into  $\sqrt{n}$  intervals of length  $d$ , of the form  $(h(q) + i \cdot d, h(q) + (i + 1)d]$ , wrapping around modulo  $t$ . One of these intervals is exactly  $Q$ .

We prescribe the distribution of keys between the intervals; the distribution within each interval will be fully random. To place  $2d = 4\sqrt{n}$  keys in the query interval with constant probability, we

mix among two strategies with constant probabilities (to be determined):

$S_1$ : Spread keys evenly, with  $\sqrt{n}$  keys in each interval.

$S_2$ : Pick the query interval  $Q$  and three random intervals. Place  $4\sqrt{n}$  keys in one of these 4 intervals, and none in the others. All other intervals get  $\sqrt{n}$  keys.

From the perspective of the stored keys, the 4 intervals are completely random. With probability  $1/4$ , it is  $Q$  that gets  $4\sqrt{n} = 2d$  keys, overloading it by a factor 2. Then, as described above, the search time is  $\Omega(\sqrt{n})$ .

To prove that the hash function is 2-independent, we need to consider pairs of two stored keys, and pairs involving the query and one stored key. In either case, we can just look at the distribution into intervals, since the position within an interval is truly random. Moreover, by symmetry between intervals, we only need to understand the probability of the two keys landing in the same interval (which we call a “collision”). We need to balance the strategies so that the collision probability is exactly  $1/\sqrt{n}$ .

Since stored keys are symmetric, the probability of  $q$  and  $x$  colliding is  $1/n$  times the expected number of items in  $Q$ , which is exactly  $\sqrt{n}$  with both strategies. Thus  $h(q)$  and  $h(x)$  are independent no matter how we mix  $S_1$  and  $S_2$ .

To analyze pairs  $(x, y)$  of stored keys, we compute the expected number of collisions among stored keys. We want this number to be  $\binom{n}{2}/\sqrt{n} = \frac{1}{2}n^{1.5} - \frac{1}{2}\sqrt{n}$ . In strategy  $S_1$ , we get the smallest possible number of collisions:  $\sqrt{n}\binom{\sqrt{n}}{2} = \frac{1}{2}n^{1.5} - \frac{1}{2}n$ . This is too few by almost  $n/2$ . In strategy  $S_2$ , we get  $(\sqrt{n} - 4)\binom{\sqrt{n}}{2} + \binom{4\sqrt{n}}{2} = \frac{1}{2}n^{1.5} + \frac{11}{2}n$  collisions, which is too much by a bit more than  $5.5n$ . To get the right expected number of collisions, we use  $S_2$  with probability  $P_{S_2} = \frac{(5.5+o(1))n}{(0.5+5.5\pm o(1))n} = \frac{11}{12} \pm o(1)$ . With this mix of strategies, our hashing of keys is 2-independent, and since  $P_{S_2} = \Omega(1)$ , our expected search cost is  $\Omega(\sqrt{n})$ .

**Upper bound** We will now prove a matching upper bound of  $O(\sqrt{n})$  on the expected query cost with any 2-independent scheme. As in the lower bound, we hash  $n$  keys into  $[t]$ ,  $t = \lceil 2n \rceil$ , and divide  $[t]$  into  $\sqrt{n}$  intervals of length  $d = 2\sqrt{n}$ . We view keys as colliding if they hash to the same interval. We want to argue that long runs imply too many collisions for 2-independence.

The expected number of collisions is  $\binom{n}{2}/\sqrt{n} = n^{3/2}/2 - n^{1/2}/2$ . The minimum number of collisions is with the distribution  $S_1$  from the lower bound: a perfectly regular distribution with  $n/\sqrt{n} = \sqrt{n}$  keys in each interval, hence  $\sqrt{n} \cdot \binom{\sqrt{n}}{2} = n^{3/2}/2 - n/2$  collisions in total.

An interval with  $m$  keys has  $m^2/2 - m/2$  collisions and the derivative is  $m - 1/2$ . It follows that if we move a key from an interval with  $m_1$  keys to one with  $m_2 \geq m_1$  keys, the number of collisions increases by more than  $m_2 - m_1$ . Any distribution can be obtained from the above minimal distribution by moving keys from intervals with at most  $\sqrt{n}$  keys to intervals with at least  $\sqrt{n}$  keys, and each such move increases the number of collisions.

A run of length  $t\sqrt{n}$  implies that this many keys hash to an interval of this length. The run is contained in less than  $t/2 + 2$  of our length  $d = 2\sqrt{n}$  intervals. In the process of creating a distribution with this run from the minimum distribution, we have to move at least  $t\sqrt{n} - 1.5\sqrt{n}(t/2 + 2)$  keys to intervals that have already been filled with at least  $1.5\sqrt{n}$  keys. Keys are always moved from intervals with less than  $\sqrt{n}$  keys, so each move gains at least  $0.5\sqrt{n}$  collisions. Thus our total gain is at least

$$0.5\sqrt{n}(t\sqrt{n} - 1.5\sqrt{n}(t/2 + 2)) = (t/8 - 1.5)n$$

The total number of collisions with a run of length  $t\sqrt{n}$  is therefore at least

$$n^{3/2}/2 - n/2 + (t/8 - 1.5)n = n^{3/2}/2 - 2n + tn/8.$$

It follows that if the expected run length is bigger than  $16\sqrt{n}$ , then the expected number of collisions is bigger than  $n^{3/2}/2$ , contradicting that their expected number is only  $n^{3/2}/2 - n^{1/2}/2$ .

## 2.2 Construction Time $\Omega(n \lg n)$ with 3-Independence

We will now construct a 3-independent family of hash functions, such that the time to insert  $n$  items into a hash table is  $\Omega(n \lg n)$ . The lower bound is based on overflowing intervals.

**Lemma 1.** *Suppose an interval  $[a, b]$  of length  $d$  has  $d + \Delta$  stored keys hashing to it. Then the insertion cost of these keys is  $\Omega(\Delta^2)$ .*

*Proof.* The overflowing  $\Delta$  keys will be part of a run containing  $(b, b + \Delta]$ . At least  $\lceil \Delta/2 \rceil$  of them must end at position  $b + \lceil \Delta/2 \rceil$  or later, i.e., a displacement of at least  $\lceil \Delta/2 \rceil$ . Interference from stored keys hashing outside  $[a, b]$  can only increase the displacement, so the insertion cost is  $\Omega(\Delta^2)$ .  $\square$

We will add up such squared overflow costs over disjoint intervals, demonstrating an expected total cost of  $\Omega(n \lg n)$ .

As before, we assume the array size  $t$  is a power of two, and we set  $n = \lceil \frac{2}{3}t \rceil$ . We imagine a perfect binary tree spanning the array. The root is level 0 and level  $\ell$  is the nodes at depth  $\ell$ . Our hash function will recursively distribute keys from a node to its two children, starting at the root. Nodes run independent random distribution processes. Then, if each node makes a  $k$ -independent distribution, overall the function is  $k$ -independent.

For a node, we mix between two strategies for distributing  $2m$  keys between the two children:

$S_1$ : Distribute the keys evenly between the children. If  $2m$  is odd, a random child gets  $\lceil m \rceil$  keys.

$S_2$ : Give all the keys to a random child.

Our goal is to determine the correct probability for the second strategy,  $P_{S_2}$ , such that the distribution process is 3-independent. Then we will calculate the cost it induces on linear probing. First, however, we need some basic facts about  $k$ -independence.

### 2.2.1 Characterizing $k$ -Independence

Our randomized procedure treats keys symmetrically, and ignores the distinction between left/right children. We call such distributions *fully symmetric*. Say the current node has to distribute  $2m$  keys to its two children ( $m$  need not be integral). Let  $X_a$  be the indicator random variable for key  $a$  ending in the left child, and  $X = \sum_a X_a$ . By symmetry of the children,  $\mathbf{E}[X_a] = \frac{1}{2}$ , so  $\mathbf{E}[X] = m$ . The  $k^{\text{th}}$  moment is  $F_k = \mathbf{E}[(X - m)^k]$ . Also define  $p_k = \Pr[X_1 = \dots = X_k = 1]$  (by symmetry, any  $k$  distinct keys yield the same value).

**Lemma 2.** *A fully symmetric distribution is  $k$ -independent iff  $p_i = 2^{-i}$  for all  $i = 2, \dots, k$ .*



*Proof.* For the non-trivial direction, assume  $p_i = 2^{-i}$  for all  $i = 2, \dots, k$ . We need to show that, for any  $(x_1, \dots, x_k) \in \{0, 1\}^k$ ,  $\Pr[(X_1 = x_1) \wedge \dots \wedge (X_k = x_k)] = 2^{-k}$ . By symmetry of the keys, we can sort the vector to  $x_1 = \dots = x_t = 1$  and  $x_{t+1} = \dots = x_k = 0$ . Let  $p_{k,t}$  be the probability that such a vector is seen.

We use induction on  $k$ . In the base case,  $p_{1,0} = p_{1,1} = \frac{1}{2}$  by symmetry. For  $k \geq 2$ , we start with  $p_{k,k} = p_k = 2^{-k}$ . We then use induction for  $t = k-1$  down to  $t = 0$ . The induction step is simply:  $p_{k,t} = p_{k-1,t} - p_{k,t+1} = 2^{-(k-1)} - 2^{-k} = 2^{-k}$ . Indeed,  $\Pr[X_{1..t} = 1 \wedge X_{t+1..k} = 0]$  can be computed as the difference between  $\Pr[X_{1..t} = 1 \wedge X_{t+1..k-1} = 0]$  (measured by  $p_{k-1,t}$ ) and  $\Pr[X_{1..t} = 1 \wedge X_{t+1..k-1} = 0 \wedge X_k = 1]$  (measured by  $p_{k,t+1}$ ).  $\square$

Based on this lemma, we can also give a characterization based on moments. First observe that any odd moment is necessarily zero, as  $\Pr[X = m + \delta] = \Pr[X = m - \delta]$  by symmetry of the children.

**Lemma 3.** *A fully symmetric distribution is  $k$ -independent iff its even moments up to  $F_k$  coincide with the moments of the truly random distribution.*

*Proof.* We will show that  $p_2, \dots, p_k$  are determined by  $F_2, \dots, F_k$ , and vice versa. Thus, any distribution that has the same moments as a truly random distribution, will have the same values  $p_2, \dots, p_k$  as the truly random distribution ( $p_i = 2^{-i}$  as in Lemma 2).

Let  $n^{\bar{k}} = n(n-1)\dots(n-k+1)$  be the falling factorial. The complete dependence between  $p_2, \dots, p_k$  and  $F_2, \dots, F_k$  follows inductively from the following statement:

$$F_k = (2m)^{\bar{k}} p_k + f_k(m, p_2, \dots, p_{k-1}), \quad \text{for some function } f_k.$$

To see this, note that  $F_k = \mathbf{E}[(X - m)^k] = \mathbf{E}[X^k] + f(m, \mathbf{E}[X^2], \dots, \mathbf{E}[X^{k-1}])$  for some function  $f$ . But  $\mathbf{E}[X^k] = (2m)^{\bar{k}} p_k + f_1(m, k) p_{k-1} + f_2(m, k) p_{k-2} + \dots$ . Here, the factors  $f_i(m, k)$  count the number of ways to select  $k$  out of  $2m$  keys, with  $i$  duplicates.  $\square$

### 2.2.2 Mixing the Strategies

As a general convention, when we are mixing strategies  $S_i$ , we use  $P_{S_i}$  to denote the probability of picking strategy  $S_i$  while we use a superscript  $S_i$  to denote measures within strategy  $S_i$ , e.g.,  $F_2^{S_i}$  is the second moment when strategy  $S_i$  is applied.

By Lemma 3, a mix of  $S_1$  and  $S_2$  is 3-independent iff it has the correct 2<sup>nd</sup> moment  $F_2 = \frac{m}{2}$ . In strategy  $S_1$ ,  $X = m \pm 1$  (due to rounding errors if  $2m$  is odd), so  $F_2^{S_1} \leq 1$ . In  $S_2$  (all to one child),  $|X - m| = m$  so  $F_2^{S_2} = m^2$ . For a correct 2<sup>nd</sup> moment of  $m/2$ , we balance with  $P_{S_2} = \frac{2}{m} \pm O(\frac{1}{m^2})$ .

### 2.2.3 The Construction Cost of Linear Probing

We now calculate the cost in terms of squared overflows. As long as the recursive steps spread the keys evenly with  $S_1$ , the load factor stays around  $2/3$ : at level  $\ell$ , the intervals have length  $2n/2^\ell$  and  $2m = 2/3 \cdot n/2^\ell \pm 1$  keys. If now, for a node  $v$  on level  $\ell$ , we apply  $S_2$  collecting all keys into one child, that child interval gets an overflow of  $1/3 \cdot n/2^\ell \pm 1 = \Omega(m)$  keys. By Lemma 1, the keys at the child will have a total insertion cost of  $\Omega(m^2)$ . Since  $P_{S_2} = \Theta(1/m)$ , the expected cost induced by  $v$  is  $\Omega(m) = \Omega(n/2^\ell)$ . This, however, assumes that no ancestor of  $v$  was collected. Note that it also avoids over-counting when we only charge  $v$  is  $S_2$  collection is applied to  $v$  but to no ancestors of  $v$ , for then the nodes charged all represent different keys.

It remains to bound the probability that  $S_2$  collection has been applied to an ancestor of a node  $v$  on a given level  $\ell \leq (\log n)/2$ . The collection probability for a node  $u$  on level  $i \leq \ell$  is  $P_{S_2} = \Theta(1/m) = \Theta(2^i/n)$  assuming no collection among the ancestors of  $u$ . By the union bound, the probability that any ancestor  $u$  of  $v$  is first to be collected is  $\sum_{i=0}^{\ell-1} \Theta(2^i/n) = \Theta(2^\ell/n) = \Theta(1/\sqrt{n}) = o(1)$ . We conclude that  $v$  has no collected ancestors with probability  $1 - o(1)$ , hence that the expected cost of  $v$  is  $\Omega(n/2^\ell)$  as above. The total expected cost over all  $2^\ell$  level  $\ell$  nodes is thus  $\Omega(n)$ . Summing over all levels  $\ell \leq (\log n)/2$ , we get an expected total insertion cost of  $\Omega(n \log n)$  for our 3-independent scheme.

### 2.3 Expected Query Time $\Omega(\lg n)$ with 4-Independence

Proving high expected search cost with 4-independence combines the ideas for 2-independence and 3-independence. However, some quite severe complications will arise. The lower bound is based on an overflowing intervals.

**Lemma 4.** *Suppose an interval  $[a, b]$  of length  $d$  has  $d + \Delta$ ,  $\Delta = \Omega(d)$ , stored keys hashing to it. Assuming that the interval has even length and that the stored keys hash symmetrically to the first and second half of  $[a, b]$ . Moreover, assume that the query key hash uniformly in  $[a, b]$ . Then the expected query time is  $\Omega(\Delta)$ .*

*Proof.* By symmetry between the first and the second half, with probability  $1/2$ , the first half gets half the keys, hence an overflow of  $\Delta/2$  keys, and a run containing  $[a + d/2, a + d/2 + \Delta/2]$ . Since  $\Delta = \Omega(d)$ , the probability that the query key hits the first half of this run is  $\Omega(1)$ , and then the expected query cost is  $\Omega(\Delta)$ .  $\square$

As for 2-independence, we will first choose  $h(q)$  and then make the stored keys cluster preferentially around  $h(q)$ . As for 3-independence, the distribution will be described using a perfectly balanced binary tree over  $[t]$ . The basic idea is to use the 3-independent distribution from Section 2.2 along the query path. For brevity, we call nodes on the query path for query nodes. The overflows that lead to an  $\Omega(n \log n)$  construction cost, will yield an  $\Omega(\log n)$  expected query time. However, the clustering of this 3-independent distribution is far too strong for 4-independence. We cannot really use it in the top of the tree, but further down, we can balance it with an anti-clustering distributions at most of the nodes outside the query path.

#### 2.3.1 3-independent Building Blocks

For a node that has  $2m$  keys to distribute, we consider three basic strategies:

$S_1$ : Distribute the keys evenly between the two children. If  $2m$  is odd, a random child gets  $\lceil m \rceil$  keys.

$S_2$ : Give all the keys to a random child.

$S_3$ : Pick a child randomly, and give it  $m + \delta = \lceil m + \sqrt{m/2} \rceil$  keys.

By mixing among these, we define two super-strategies:

$$T_1 = P_{S_2} \times S_2 + (1 - P_{S_2}) \times S_1;$$

$$T_2 = P_{S_3} \times S_3 + (1 - P_{S_3}) \times S_1.$$

The above notation means that strategy  $T_1$  picks strategy  $S_2$  with probability  $P_{S_2}$ ;  $S_1$  otherwise. Likewise  $T_2$  picks  $S_3$  with probability  $P_{S_3}$ ;  $S_1$  otherwise. The probabilities  $P_{S_2}$  and  $P_{S_3}$  are chosen such that  $T_1$  and  $T_2$  are 3-independent. The strategy  $T_1$  is the 3-independent strategy from Section 2.2 where we determined  $P_{S_2} = \frac{2}{m} \pm O(\frac{1}{m^2})$ . This will be our preferred strategy on the query path.

To compute  $P_{S_3}$ , we employ the 2<sup>nd</sup> moments:  $F_2^{S_1} \leq 1$  and  $F_2^{S_3} = \frac{m}{2} + O(\sqrt{m})$ . (If one ignored rounding, we would have the precise bounds  $F_2^{S_1} = 0$  and  $F_2^{S_3} = \frac{m}{2}$ .) By Lemma 3, we need a 2<sup>nd</sup> moment of  $m/2$ . Thus, we have  $P_{S_3} = 1 - O(\frac{1}{\sqrt{m}})$ .

### 2.3.2 4-Independence on the Average, One Level At The Time

We are going to get 4-independence by an appropriate mix of our 3-independent strategies  $T_1$  and  $T_2$ . Our first step is to hash the query uniformly into  $[t]$ . This defines the query path. We will do the mixing top-down, one level  $\ell$  at the time. The individual node will not distribute its keys 4-independently. Nodes on the query path will prefer  $T_1$  while keys outside the query path will prefer  $T_2$ , all in a mix that leads to global 4-independence. There will also be neutral nodes for which we use a truly random distribution. Since all distributions are 3-independent regardless of the query path, the query hashes independently of any 3 stored keys. We are therefore only concerned about the 4-independence among stored keys.

It is tempting to try balancing of  $T_1$  and  $T_2$  via 4<sup>th</sup> moments using Lemma 3. However, even on the same level  $\ell$ , the distribution of the number of keys at the node on the query path will be different from the distributions outside the query path, and this makes balancing via 4<sup>th</sup> moments non-obvious. Instead, we will argue independence via Lemma 2: since we already have 3-independence and all distributions are symmetric, we only need to show  $p_4 = 2^{-4}$ . Thus, conditioned on 4 given keys  $a, b, c, d$  being together on level  $\ell$ , we want them all to go to the left child with probability  $2^{-4}$ . By symmetry, our 4-tuple  $(a, b, c, d)$  is uniformly random among all 4-tuples surviving together on level  $\ell$ . On the average we thus want such 4-tuples to go left together with probability  $2^{-4}$ .

### 2.3.3 Analyzing $T_1$ and $T_2$

Our aim now is to compute  $p_4^{T_1}$  and  $p_4^{T_2}$  for a node with  $2m$  keys to be split between its children.

First we note:

$$p_4^{S_1} = m^4 / (2m)^4 = \frac{1}{2^4} \left(1 - \frac{6}{m} \pm \frac{O(1)}{m^2}\right)$$

Indeed, the first key will go to the left child with probability  $\frac{1}{2} = \frac{m}{2m}$ . Conditioned on this, the second key will go to the left child with probability  $\frac{m-1}{2m}$ , etc. In  $S_2$ , all keys go to the left child with probability a half, so  $p_4^{S_2} = \frac{1}{2}$ . Since  $P_{S_2} = \frac{2}{m} \pm O(\frac{1}{m^2})$ , we get

$$p_4^{T_1} = P_{S_2} \cdot p_4^{S_2} + (1 - P_{S_2}) p_4^{S_1} = \frac{1}{2^4} \left(1 + \frac{8}{m} \pm \frac{O(1)}{m^2}\right) = 2^{-4} + \Theta(1/m).$$

To avoid a rather involved calculation, we will not derive  $p_4^{T_2}$  directly, but rather as a function of the 4<sup>th</sup> moment. We have  $F_4^{S_1} \leq 1$ ,  $F_4^{S_3} = \delta^4 = \frac{1}{4}m^2 + O(m^{3/2})$ , and  $P_{S_3} = 1 - O(\frac{1}{\sqrt{m}})$ , so

$$F_4^{T_2} = P_{S_3} F_4^{S_3} + (1 - P_{S_3}) F_4^{S_1} = \frac{1}{4}m^2 \pm O(m^{3/2}).$$

From the proof of Lemma 3, we know that  $F_4 = (2m)^4 p_4 + f_k(m, p_2, p_3)$  with any distribution. Since  $T_2$  is 3-independent, it has the same  $p_2$  and  $p_3$  as a truly random distribution. Thus, we can

compute  $f(m, p_2, p_3)$  using the  $p_4$  and  $F_4$  values of a truly random distribution. The 4<sup>th</sup> moment of a truly random distribution is:

$$F_4 = \frac{2m}{2^4} + \binom{4}{2} \frac{(2m)^2}{2^4} = \frac{24m^2 - 10m}{2^4}.$$

Since  $p_4 = 2^{-4}$  in the truly random case, we have:  $f(m, p_2, p_3) = 2^{-4}[(2m)^4 - (24m^2 - 10m)]$ . Now we can return to  $p_4^{T_2}$ :

$$\begin{aligned} p_4^{T_2} &= \frac{F_4^{T_2} + f(m, p_2, p_3)}{(2m)^4} = \frac{1}{2^4} \left( \frac{4m^2 \pm O(m^{3/2})}{(2m)^4} + 1 - \frac{24m^2 - 10m}{(2m)^4} \right) \\ &= \frac{1}{2^4} \left( 1 - \frac{20m^2 \pm O(m^{1.5})}{(2m)^4} \right) = \frac{1}{2^4} \left( 1 - \frac{20}{m^2} \pm \frac{O(1)}{m^{2.5}} \right) = 2^{-4} - \Theta(1/m^2). \end{aligned}$$

To get  $p_4 = 2^{-4}$  for a given node, we use a strategy  $T^*$  that applies  $T_1$  with probability  $P_{T_1}^* \approx 5/(2m)$ ;  $T_2$  otherwise. However, as stated earlier, we will often give preference to  $T_1$  on the query path, and to  $T_2$  elsewhere.

### 2.3.4 The Distribution Tree

We are now ready to describe the mix of strategies used in the binary tree. On the top  $\frac{2}{3} \log_2 t$  levels, we use the above mentioned mix  $T^*$  of  $T_1$  and  $T_2$  yielding a perfect 4-independent distribution of the keys at each node.

On the next levels  $\ell \geq \frac{2}{3} \log_2 t$ , we will always use  $T_1$  on the query path. For the other nodes, we use  $T_1$  with the probability  $P_{T_1}^-$  such that if all non-query nodes on level  $\ell$  use the strategy

$$T^- = P_{T_1}^- \times T_1 + (1 - P_{T_1}^-) \times T_2;$$

then we get  $p_4 = 2^{-4}$  for an average 4-tuple on level  $\ell$ . We note that  $P_{T_1}^-$  depends completely on the distribution of 4-tuples at the nodes on level  $\ell$  and that  $P_{T_1}^-$  has to compensate for the fact that  $T_1$  is used at the query node. We shall prove the existence of  $P_{T_1}^-$  shortly.

Finally, we have a stopping criteria: if at some level  $\ell$ , we use the  $S_2$  collection on the query path, or if  $\ell + 1 > \frac{5}{6} \log_2 t$ , then we use a truly random distribution on all subsequent levels. We note that the  $S_2$  collection could happen already on a top level  $\ell \leq \frac{2}{3} \log_2 t$ .

### 2.3.5 Possibility of Balance

Consider a level  $\ell$  before the stopping criteria has been applied. We need to argue that the above mentioned probability  $P_{T_1}^-$  exists. We will argue that  $P_{T_1}^- = 0$  implies  $p_4 < 2^{-4}$  while  $P_{T_1}^- = 1$  implies  $p_4 > 2^{-4}$ . Then continuity implies that there exists a  $P_{T_1}^- \in [0, 1]$  yielding  $p_4 = 2^{-4}$ .

With  $P_{T_1}^- = 1$ , we use strategy  $T_1$  for all nodes on the level, and we already know that  $p_4^{T_1} > 2^{-4}$ .

Now consider  $P_{T_1}^- = 0$ , that is, we use  $T_1$  only at the query node. Starting with a simplistic calculation, assume that all  $2^\ell$  nodes on level  $\ell$  had exactly  $2m = n/2^\ell$  keys, hence the same number of 4-tuples. Then the average is

$$\frac{p_4^{T_1} + (2^\ell - 1)p_4^{T_2}}{2^\ell} = \frac{2^{-4} + \Theta(1/m) - (2^\ell - 1)(2^{-4} + \Theta(1/m^2))}{2^\ell} < 2^{-4}.$$

The inequality follows because  $\ell \geq \frac{2}{3} \log_2 t$  implies  $2^\ell > n^{2/3}$  while  $m < n/2^\ell \leq n^{1/3}$ . However, the number of keys at different nodes on level  $\ell$  is not expected to be the same, and we will handle this below.

We want to prove that the average  $p_4$  over all 4-tuples on level  $\ell$  is below  $2^{-4}$ . To simplify calculations, we can add  $p_4^{T_1} - 2^{-4} = \Theta(1/m)$  for each 4-tuple using  $T_1$  and  $p_4^{T_2} - 2^{-4} = -\Theta(1/m^2)$  for each tuple using  $T_2$ , and show that the sum is negative. If the query node has  $2m$  keys, all using  $T_1$ , we thus add  $(2m)^4 \Theta(1/m) = \Theta(m^3)$ . If a non-query node has  $2m$  keys, we subtract  $(2m)^4 \Theta(1/m^2) = \Theta(m^2)$ .

We now want to bound the number of keys at the level  $\ell$  query node. Since the stopping criteria has not applied, we know that  $S_2$  collection has not been applied to any of its ancestors.

**Lemma 5.** *If we have never applied  $S_2$  collection on the path to a query node  $v$  on level  $j \leq \frac{5}{6} \log_2 t$ , then  $v$  has  $n/2^j \pm 3\sqrt{n/2^j}$  keys.*

*Proof.* On the path to  $v$ , we have only applied strategies  $S_1$  and  $S_3$ . Hence, if an ancestor of  $v$  has  $2m$  keys, then each child gets  $m \pm (\sqrt{m/2} + 1)$  keys. The bound follows by induction starting with  $2m = n$  keys at the root on level 0.  $\square$

Our level  $\ell$  query node thus has  $\Theta(n/2^\ell)$  keys and contributes  $O((n/2^\ell)^3)$  to the sum.

To lower bound the negative contribution from the non-query nodes on level  $\ell$ , we first note that they share all the  $n - O(n/2^\ell) = \Omega(n)$  keys not on the query path. The negative contribution for a node with  $2m$  keys is  $\Omega(m^2)$ . By convexity, the total negative contribution is minimized if the keys are evenly spread among the  $2^\ell - 1$  non-query nodes, and even less if we distributed on  $2^\ell$  nodes. The total negative contribution is therefore at least  $2^\ell \Omega((n/2^\ell)^2) = \Omega(n^2/2^\ell)$ . This dominates the positive contribution from the query node since  $(2^\ell)^2 \geq n^{4/3} = \omega(n)$ . Thus we conclude that  $p_4 < 2^{-4}$  when  $P_{T_1}^- = 0$ . This completes the proof that we for level  $\ell$  can find a value of  $P_{T_1}^- \in [0, 1]$  such that  $p_4 = 2^{-4}$ , hence the proof that the distribution tree described in Section 2.3.4 exists, hashing all keys 4-independently.

### 2.3.6 Expected Query Time

We will now study the expected query time. We only consider the cost in the event that  $S_2$  collection is applied at the query node at some level  $\ell \in [\frac{2}{3} \log_2 t, \frac{5}{6} \log_2 t]$ . This implies that  $S_2$  has not been applied previously on the query path, so the event can only happen once with a given distribution (no over counting). By Lemma 5, our query node has  $n/2^\ell \pm 3\sqrt{n/2^\ell}$  keys. With probability  $1/2$ , these all go to the query child which represents an interval of length  $t/2^{\ell+1}$ . Since  $n = 2t/3$ , we conclude that the query child gets overloaded by almost a factor  $4/3$ . By Lemma 4, the expected search cost is then  $\Omega(n/2^\ell)$ .

On the query path on every level  $i \leq \ell$ , we know that the probability of applying  $S_2$  provided that  $S_2$  has not already been applied is  $\Theta(1/m)$  where  $m = \Theta(n/2^i)$  by Lemma 5. The probability of applying  $S_2$  on level  $\ell \in [\frac{2}{3} \log_2 t, \frac{5}{6} \log_2 t]$  is therefore  $(1 - \sum_{i=0}^{\ell-1} O(2^i/n))\Theta(2^\ell/n) = \Theta(2^\ell/n)$ , so the expected search cost from this level is  $\Theta(1)$ . Since our event can only happen on one level for a given distribution, we sum this cost over the  $\Omega(\log n)$  levels in  $[\frac{2}{3} \log_2 t, \frac{5}{6} \log_2 t]$ . We conclude that the expected search cost of our 4-independent scheme is  $\Omega(\log n)$ .

### 3 Minwise Independence via $k$ -Independence

We will show that it is limited how good minwise independence we can achieve based on  $k$ -independent hashing. For a given  $k$ , our goal is to construct a  $k$ -independent distribution over  $n$  regular keys and a query key  $q$ , such that the probability that  $q$  gets the minimal hash value is  $\frac{1}{n+1}(1 + 2^{-O(k)})$ .

We assume that  $k$  is even and divides  $n$ . Each hash value will be uniformly distributed in the unit interval  $[0, 1)$ . Discretizing this continuous interval does not affect any of the calculations below, as long as precision  $2 \lg n$  or more is used (making the probability of a non-unique minimum vanishingly small).

For our construction, we divide the unit interval into  $\frac{n}{k}$  subintervals of the form  $[i \frac{k}{n}, (i+1) \frac{k}{n})$ . The regular keys are distributed totally randomly between these subintervals. Each subinterval  $I$  gets  $k$  regular keys in expectation. We say that  $I$  is *exact* if it gets exactly  $k$  regular keys. Whenever  $I$  is not exact, the regular keys are placed totally randomly within it.

The distribution inside an exact interval  $I$  is dictated by a parity parameter  $P \in \{0, 1\}$ . We break  $I$  into two equal halves, and distribute the  $k$  keys into these halves randomly, conditioned on the parity in the first half being  $P$ . Within its half, each key gets an independent random value. If  $P$  is fixed, this process is  $k-1$  independent. Indeed, one can always deduce the half of a key  $x$  based on knowledge of  $k-1$  keys, but the location of  $x$  is totally uniform if we only know about  $k-2$  keys. If the parity parameter  $P$  is uniform in  $\{0, 1\}$  (but possibly dependent among exact intervals), the overall distribution is still  $k$ -independent.

The query is generated independently and uniformly. For each exact interval  $I$ , if the query is inside it, we set its parity parameter  $P_I = 0$ . If  $I$  is exact but the query is outside it, we toss a biased coin to determine the parity, with  $\Pr[P_I = 0] = (\frac{1}{2} - \frac{k}{n}) / (1 - \frac{k}{n})$ . Any fixed exact interval receives the query with probability  $\frac{k}{n}$ , so overall the distribution of  $P_I$  is uniform.

We claim that the overall process is  $k$ -independent. Uniformity of  $P_I$  implies that the distribution of regular keys is  $k$ -independent. In the case of  $q$  and  $k-1$  regular keys, we also have full independence, since the distribution in an interval is  $(k-1)$ -independent even conditioned on  $P$ .

It remains to calculate the probability of  $q$  being the minimum under this distribution. First we assume that the query landed in an exact interval  $I$ , and calculate  $p_{\min}$ , the probability that  $q$  takes the minimum value within  $I$ . Define the random variable  $X$  as the number of regular keys in the first half. By our process,  $X$  is always even.

If  $X = x > 0$ ,  $q$  is the minimum only if it lands in the first half (probability  $\frac{1}{2}$ ) and is smaller than the  $x$  keys already there (probability  $\frac{1}{x+1}$ ). If  $X = 0$ ,  $q$  is the minimum either if it lands in the first half (probability  $\frac{1}{2}$ ), or if it lands in the second half, but is smaller than everybody there (probability  $\frac{1}{2(k+1)}$ ). Thus,

$$p_{\min} = \Pr[X = 0] \cdot \left(\frac{1}{2} + \frac{1}{2(k+1)}\right) + \sum_{x=2,4,\dots,k} \Pr[X = x] \cdot \frac{1}{2(x+1)}$$

To compute  $\Pr[X = x]$ , we can think of the distribution into halves as a two step process: first  $k-1$  keys are distributed randomly; then, the last key is placed to make the parity of the first half even. Thus,  $X = x$  if either  $x$  or  $x-1$  of the first  $k-1$  keys landed in the first half. In other words:

$$\Pr[X = x] = \binom{k-1}{x} / 2^{k-1} + \binom{k-1}{x-1} / 2^{k-1} = \binom{k}{x} / 2^{k-1}$$

No keys are placed in the first half iff none of the first  $k - 1$  keys land there; thus  $\Pr[X = 0] = 1/2^{k-1}$ . We obtain:

$$p_{\min} = \frac{1}{2^k(k+1)} + \frac{1}{2^k} \sum_{x=0,2,\dots,k} \frac{1}{x+1} \binom{k}{x}$$

But  $\frac{1}{x+1} \binom{k}{x} = \frac{1}{k+1} \binom{k+1}{x+1}$ . Since  $k+1$  is odd, the sum over all odd binomial coefficients is exactly  $2^{k+1}/2$  (it is equal to the sum over even binomial coefficients, and half the total). Thus,  $p_{\min} = \frac{1}{2^k(k+1)} + \frac{1}{k+1}$ , i.e.  $q$  is the minimum with a probability that is too large by a factor of  $1 + 2^{-k}$ .

We are now almost done. For  $q$  to be the minimum of all keys, it has to be in the minimum non-empty interval. If this interval is exact, our distribution increases the chance that  $q$  is minimum by a factor  $1 + 2^{-k}$ ; otherwise, our distribution is completely random in the interval, so  $q$  is minimum with its fair probability. Let  $Z$  be the number of regular keys in  $q$ 's interval, and let  $\mathcal{E}$  be the event that  $q$ 's interval is the minimum non-empty interval. If the distribution were truly random, then  $q$  would be minimum with probability:

$$\frac{1}{n+1} = \sum_z \Pr[Z = z] \cdot \Pr[\mathcal{E} \mid Z = z] \cdot \frac{1}{z+1}$$

In our tweaked distribution,  $q$  is minimum with probability:

$$\begin{aligned} & \sum_{z \neq k} \Pr[Z = z] \cdot \Pr[\mathcal{E} \mid Z = z] \cdot \frac{1}{z+1} + \Pr[Z = k] \cdot \Pr[\mathcal{E} \mid Z = k] \cdot \frac{1 + 2^{-k}}{k+1} \\ &= \frac{1}{n+1} + \Pr[Z = k] \cdot \Pr[\mathcal{E} \mid Z = k] \cdot \frac{2^{-k}}{k+1} \end{aligned}$$

But  $Z$  is a binomial distribution with  $n$  trials and mean  $k$ ; thus  $\Pr[Z = k] = \Omega(1/\sqrt{k})$ . Furthermore,  $\Pr[\mathcal{E} \mid Z = k] \geq \frac{k}{n}$ , since  $q$ 's interval is the very first with probability  $\frac{k}{n}$  (and there is also a nonzero chance that it is not the first, but all interval before are empty). Thus, the probability is off by an additive term  $\frac{\Omega(2^{-k}/\sqrt{k})}{n}$ . This translates into a multiplicative factor of  $1 + 2^{-O(k)}$ .

## 4 Multiply-Shift and Linear Probing

We show that the simplest and fastest known universal hashing schemes have bad expected performance when used for linear probing on some of the most realistic structured data. This result is inspired by negative experimental findings from [22]. The essential form of the schemes considered have the following basic form: we want to hash  $\ell_{in}$ -bit keys into  $\ell_{out}$ -bit indices. Here  $\ell_{in} \geq \ell_{out}$ , and the indices are used for the linear probing array. For the typical case of a half full table, we have  $2^{\ell_{out}} = m \approx 2n$ . In particular,  $m > n$ .

Depending on details of the scheme, for some  $\ell \geq \ell_{in}, \ell_{out}$ , we pick a random multiplier  $a \in [2^\ell]$ , and compute

$$h_a(x) = (ax \bmod 2^\ell) \div 2^{\ell - \ell_{out}}$$

We refer to this as the *basic multiply-shift scheme*. If  $\ell \in \{8, 16, 32, 64\}$ , the mod-operation is performed automatically as discarded overflow. The  $\div$  operation is just a right shift by  $s = \ell - \ell_{out}$ , so in C we get the simple code `(a*x)>>s` and the cost is dominated by a single multiplication. For

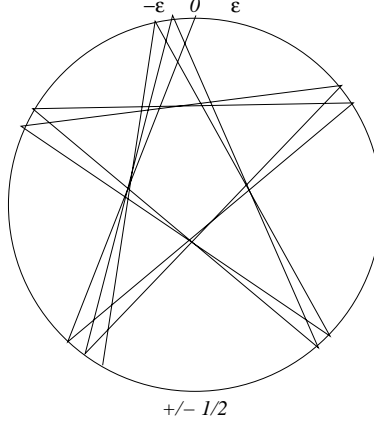


Figure 1: Case where  $h_a^0(5) \leq \varepsilon$ .

the plain universal hashing in [7], it suffices that  $\ell \geq \ell_{in}$  but then the multiplier  $a$  should be odd. For 2-independent hashing as in [6], we need  $\ell \geq \ell_{in} + \ell_{out} - 1$ . Also we need to add a random number  $b$ , but as we shall discuss in the end, these details have no essential impact on the derivation below.

Our basic bad example will be where the keys form the interval  $[n] = \{0, \dots, n-1\}$ . However, the problem will not go away if this interval is shifted or not totally full, or replaced by an arithmetic progression.

When analyzing the scheme, it is convenient to first consider it as a mapping into the unit interval  $[0, 1)$  via

$$h_a^0(x) = (ax \bmod 2^\ell) / 2^\ell.$$

Then  $h_a(x) = \lfloor h_a^0(x) 2^{\ell_{out}} \rfloor$ . We think of the unit interval as circular, and for any  $x \in [0, 1)$ , we define

$$\|x\| = \min\{x \bmod 1, -x \bmod 1\}.$$

This is the distance from 0 in the circular unit interval.

**Lemma 6.** *Let the multiplier  $a$  be given and suppose for some  $x \in \{1, \dots, n-1\}$  that  $\|h_a^0(x)\| \leq 1/(2m)$ . Then, when we use  $h_a$  to hash  $[n]$  into a linear probing table, the average cost per key is  $\Omega(n/x)$ .*

*Proof.* The case studied is illustrated in Figure 1. For each  $k \in [x]$ , consider the set  $[n]_k^x = \{y \in [n] \mid y = k \pmod{x}\}$ . The  $q \approx n/x$  keys from  $[n]_k^x$  map to an interval of length  $(q-1)/(2m)$ , which means that the  $h_a$  distributes  $[n]_k^x$  on at most  $\lceil q/2 \rceil + 1$  consecutive array locations. Linear probing will have to spread  $[n]_k^x$  on  $q$  locations, so on the average, the keys will get a displacement of  $\Omega(q) = \Omega(n/x)$ . We get a corresponding double full interval for every equivalence class modulo  $x$ . Therefore we get an average insert cost of  $\Omega(n/x)$  over all the keys. The above average costs only measures the interaction between keys from the same equivalence class modulo  $x$ . If some of these classes overlapped, the cost would only grow.  $\square$

Note that  $\|h_a^0(x)\| \leq 1/(2m)$  implies that  $h_a^0(x)$  is contained in an interval of size  $1/m$  around 0. From the universality arguments of [7, 6] we know that the probability of this event is roughly  $1/m$  (we shall return with an exact statement and proof later). We would like to conclude that



the expected average cost is  $\sum_{x=1}^n \Omega(n/x)/m = \Omega(\lg n)$ . The answer is correct, but the calculation cheats in the sense that we may have many different  $x$  such that  $\|h_a^0(x)\| \leq 1/(2m)$ , and the associated costs should not all be added up.

To get a proper lower bound, for any given multiplier  $a$ , we let  $\mu_a$  denote the minimal positive value such that  $\|h_a^0(\mu_a)\| \leq 1/(2m)$ . If  $\mu_a < n$ , then by Lemma 6, the average insertion cost is  $\Omega(n/\mu_a)$ . Therefore, if  $a$  is random over some probability distribution (to be played with as we go along), the expected average insert cost is lower bounded by

$$\Omega \left( \sum_{x=1}^n n/x \cdot \Pr_a[x = \mu_a] \right). \quad (1)$$

**Lemma 7.** *For a given multiplier  $a$ , consider any  $x < n$  such that  $\|h_a^0(x)\| \leq 1/(2m)$ . Then  $x \neq \mu_a$  if and only if for some prime factor  $p$  of  $x$ ,  $\|h_a^0(x/p)\| \leq 1/(2pm)$ .*

*Proof.* The “if” part is trivial. Since  $\|h_a^0(\mu_a)\| \leq 1/(2m)$ , for any integer  $i < m$ , we have  $\|h_a^0(i\mu_a)\| = i\|h_a^0(\mu_a)\|$ . Hence  $\|h_a^0(i\mu_a)\| \leq 1/(2m) \iff \|h_a^0(\mu_a)\| \leq 1/(2im)$ . On the other hand, suppose  $\|h_a^0(y)\| \leq 1/(2m)$  where  $y$  is not a multiple of  $\mu_a$ . Then  $h_a^0$  maps  $\{0, \dots, y + \mu_a - 1\}$  to points in the cyclic unit interval that are at most  $1/(2m)$  apart (c.f. Figure 1). It follows that  $y \geq 2m - \mu_a$ . However, we are only considering  $x < n$  with  $\|h_a^0(x)\| \leq 1/(2m)$ . If  $x \neq \mu_a$ , then  $\mu_a < x < n < m$  while  $y \geq 2m - \mu_a > n$ , so we conclude that  $x$  is a multiple of  $\mu_a$ . Then  $x = i\mu_a$  where  $\|h_a^0(\mu_a)\| \leq 1/(2im)$ . Let  $p$  be any prime factor of  $i$ . Then  $\|h_a^0(x/p)\| = \|h_a^0(i/p\mu_a)\| = i/p\|h_a^0(\mu_a)\| \leq 1/(2pm)$ .  $\square$

To illustrate the basic accounting idea, assume for simplicity that we have a perfect distribution  $\mathcal{U}$  on  $a$  that for any fixed  $x > 0$  distributes  $h_a^0(x)$  uniformly in the unit interval. Then for any  $x$  and  $\varepsilon < 1/2$ ,

$$\Pr_{a \leftarrow \mathcal{U}}[\|h_a^0(x)\| \leq \varepsilon] = 2\varepsilon. \quad (2)$$

Then by Lemma 7,

$$\begin{aligned} \Pr_{a \leftarrow \mathcal{U}}[x = \mu_a] &\geq \Pr_{a \leftarrow \mathcal{U}}[\|h_a^0(x)\| \leq 1/(2m)] - \sum_{p \text{ prime factor of } x} \Pr_{a \leftarrow \mathcal{U}}[\|h_a^0(x/p)\| \leq 1/(2pm)] \\ &= 1/m - \sum_{p \text{ prime factor of } x} 1/(pm) \\ &= \left( 1 - \sum_{p \text{ prime factor of } x} 1/p \right) / m \end{aligned} \quad (3)$$

We note that the lower-bound (3) may be negative since there are values of  $x$  for which  $\sum_{p \text{ prime factor of } x} 1/p = \Theta(\lg \lg x)$ . Nevertheless (3) suffices with an appropriate reordering of terms. From (1) we get that the expected average insertion cost is:

$$\begin{aligned} \Omega \left( \sum_{x=1}^n n/x \cdot \Pr_{a \leftarrow \mathcal{U}}[x = \mu_a] \right) &= \Omega \left( \sum_{x=1}^n \left( n/(xm) \left( 1 - \sum_{p \text{ prime factor of } x} 1/p \right) \right) \right) \\ &= \Omega \left( \sum_{x=1}^n \left( n/(xm) \left( 1 - \sum_{p=2,3,5,\dots} 1/p^2 \right) \right) \right) \end{aligned}$$

Above we simply moved terms of the form  $-n/(xmp)$  where  $p$  is a prime factor of  $x$  to  $x' = x/p$  in the form  $-n/(x'mp^2)$ . Conservatively, we include  $-n/(x'mp^2)$  for all primes  $p$  even if  $px' > n$ . Since  $\sum_{\text{prime } p=2,3,5,\dots} 1/p^2 < 0.453$ , we get an expected average insertion cost of

$$\begin{aligned} \Omega \left( \sum_{x=1}^n n/x \cdot \Pr_{a \leftarrow \mathcal{U}}[x = \mu_a] \right) &= \Omega \left( \sum_{x=1}^n 0.547n/(xm) \right) \\ &= \Omega(n/m \lg n). \end{aligned}$$

We would now be done if we had the perfect distribution  $\mathcal{U}$  on  $a$  so that the equality (2) was satisfied. Instead we will use the weaker statements of the following lemma:

**Lemma 8.** *Let  $\mathcal{O}$  the uniform distribution on odd  $\ell$ -bit numbers. For any odd  $x < n$  and  $\varepsilon < 1/2$ ,*

$$\Pr_{a \leftarrow \mathcal{O}}[\|h_a^0(x)\| \leq \varepsilon] \leq 4\varepsilon \quad (4)$$

*However, if  $\varepsilon$  is an integer multiple  $1/2^{\ell-1}$ , then*

$$\Pr_{a \leftarrow \mathcal{O}}[\|h_a^0(x)\| \leq \varepsilon] = 2\varepsilon. \quad (5)$$

*Proof.* When  $x$  is odd and  $a$  is a uniformly distributed odd  $\ell$ -bit number, then  $ax \bmod 2^\ell$  is uniformly distributed odd  $\ell$ -bit number. To get  $h_a^0(x)$ , we divide by  $2^\ell$ , and then we have a uniform distribution on odd multiples of  $1/2^\ell$ . Now (6) is immediate because we have exactly one odd multiple in each interval  $[i/2^{\ell-1}, (i+1)/2^{\ell-1}]$ . Also, we maximize  $\Pr_{a \leftarrow \mathcal{O}}[\|h_a^0(x)\| \leq \varepsilon]/\varepsilon$  when we capture the points closest to 0 with  $\varepsilon = 1/2^\ell$ , that is, with  $\Pr_{a \leftarrow \mathcal{O}}[\|h_a^0(x)\| \leq 1/2^\ell] = 4/2^\ell$ . Hence (4) follows.  $\square$

We are now ready to prove our lower bound for the performance of linear probing with the basic multiply-shift scheme.

**Theorem 9.** *Suppose  $\ell_{\text{out}} < \ell$  and that the multiplier  $a$  is a uniformly distributed odd  $\ell$ -bit number. If we use  $h_a$  to insert  $[n]$  in a linear probing table, then expected average insertion cost is  $\Omega(\log n)$ .*

*Proof.* By assumption  $1/m = 1/2^{\ell_{\text{out}}}$  is a multiple of  $1/2^{\ell-1}$ , so for odd  $x < n$ , (5) implies

$$\Pr_{a \leftarrow \mathcal{O}}[\|h_a^0(x)\| \leq 1/m] = 2/m. \quad (6)$$

By Lemma 7 combined with (4) and (6), we get that

$$\begin{aligned} \Pr_{a \leftarrow \mathcal{U}}[x = \mu_a] &\geq \Pr_{a \leftarrow \mathcal{U}}[\|h_a^0(x)\| \leq 1/(2m)] - \sum_{p \text{ prime factor of } x} \Pr_{a \leftarrow \mathcal{U}}[\|h_a^0(x/p)\| \leq 1/(2pm)] \\ &\geq 1/m - \sum_{p \text{ prime factor of } x} 2/(pm) \end{aligned}$$

From (1) we get that the expected average insertion cost is:

$$\begin{aligned}
\Omega\left(\sum_{x=1}^n n/x \cdot \Pr_{a \leftarrow \mathcal{U}}[x = \mu_a]\right) &= \Omega\left(\sum_{x=1}^n \left(n/(xm) \left(1 - 2 \sum_{\text{prime factor } p \text{ of } x} 1/p\right)\right)\right) \\
&= \Omega\left(\sum_{x=1}^n \left(n/(xm) \left(1 - 2 \sum_{\text{prime } p=3,5,\dots} 1/p^2\right)\right)\right) \\
&= \Omega\left(\sum_{\text{odd } x=1}^n 0.594n/(xm)\right) \\
&= \Omega(n/m \lg n).
\end{aligned}$$

Above we again moved terms of the form  $-n/(xmp)$  where  $p$  is a prime factor of  $x$  to  $x' = x/p$  in the form  $-n/(x'mp^2)$ . Since  $x$  is odd, we only have to consider odd primes factors  $p$ , and then we used that  $\sum_{\text{prime } p=3,5,\dots} 1/p^2 < 0.203$ . This completes the proof of Theorem 9.  $\square$

We note that the plain universal hashing from [7] also assumes an odd multiplier, so Theorem 9 applies directly if  $\ell_{out} < \ell$ . The condition  $\ell_{out} < \ell$  is, in fact, necessary for bad performance. If  $\ell_{out} = \ell$ , then  $h_a$  is a permutation for any odd  $a$ , and then linear probing works perfectly.

For the 2-universal hashing [6] there are two differences. One is that the multiplier may also be even, but restricting it to be odd can only double the cost. The other difference is that we add an additional  $\ell$ -bit parameter  $b$ , yielding a scheme of the form:

$$h_{a,b}(x) = \lfloor ((ax + b) \bmod 2^\ell) / 2^{\ell - \ell_{out}} \rfloor.$$

The only effect of  $b$  is a cyclic shift of the double full buckets, and this has no effect on the linear probing cost. For the 2-independent hashing, we have  $\ell \geq \ell_{in} + \ell_{out} - 1$ , so  $\ell < \ell_{out}$  if  $\ell_{in} > 1$ . Hence again we have an expected average linear probing cost of  $\Omega(n/m \lg n)$ .

Finally, we sketch some variations of our bad input. Currently, we just considered the set  $[n]$  of input keys, but it makes no essential difference if instead for some integer constants  $\alpha$  and  $\beta$ , we consider the arithmetic sequence  $\alpha[n] + \beta = \{i\alpha + \beta \mid x \in [n]\}$ . The  $\beta$  is just adds a cyclic shift like the  $b$  in 2-independent hashing. If  $\alpha$  is odd, then it is absorbed in the random multiplier  $a$ . What we get now is that if for some  $x \in [n]$ , we have  $\|h_a^0(\alpha x)\| \leq 1/(2m)$ , then again we get an average cost  $\Omega(n/x)$ . A consequence is that no odd multiplier  $a$  is universally safe because there always exists an inverse  $\alpha$  (with  $a\alpha \bmod 2^\ell = 1$ ) leading to a linear cost if  $h_a$  is used to insert  $\alpha[n] + \beta$ . It not hard to also construct bad examples for even  $\alpha$ . If  $\alpha$  is an odd multiple of  $2^i$ , we just have to strengthen the condition  $\ell_{out} < \ell$  to  $\ell_{out} < \ell - i$  to get the expected average insertion cost of  $\Omega(n/m \lg n)$ . This kind of arithmetic sequences could be a true practical problem. For example, in some denial-of-service attacks, one often just change some bits in the middle of a header key, and this gives an arithmetic sequence.

Another more practical concern is if the input set  $X$  is an  $\varepsilon$ -fraction of  $[n]$ . As long as  $\varepsilon > 2/3$ , the above proof works almost unchanged. For smaller  $\varepsilon$ , our bad case is if  $\|h_a^0(x)\| \leq \varepsilon/(2m)$ . In that case, for each  $k \in [x]$ , the  $q = \lfloor n/x \rfloor$  potential keys  $y$  from  $[n]$  with  $y \bmod x = k$  would map to an interval of length  $\varepsilon(q-1)/(2m)$ . This means that  $h_a$  spreads these potential keys on at most  $\lceil \varepsilon q/2 \rceil + 1$  consecutive array locations. A  $\varepsilon$ -fraction of these keys are real, so on the average, these intervals become double full, leading to an average cost of  $\Omega(\varepsilon n/x)$ . Strengthening  $\ell_{out} < \ell$  to

$\varepsilon \geq 2^{\ell_{out}-\ell}$ , we essentially get that all probabilities are reduced by  $\varepsilon$ . Thus we end with a cost of  $\Omega(\varepsilon^2 n / m \lg n) = \Omega(\varepsilon |X| / m \lg n)$ .

## 5 Multiplication-Shift and Minwise Independence

We now consider the lack of minwise independence with a hashing scheme

$$h_{a,b}(x) = ((ax + b) \bmod 2^\ell).$$

Shifting out less significant bits does not make much sense since we are not hashing to entries in an array. The analysis will be very similar to the one done for linear probing in Section 4, and we will only sketch it. Now the added  $b$  is necessary to get anything meaningful, for without it, zero would always get the minimal hash value  $h_{a,0}(0) = 0$ . The effect of adding  $b$  is to randomly spin the wheel from Figure 1. As in Section 4, it is convenient to divide by  $2^\ell$  to get fractions in the cyclic unit interval. We thus define  $h_{a,b}^0(x) = h_{a,b}(x)/2^\ell$ .

The bad case will be the interval  $[n]$  versus a random query key  $q$ . We assume that  $n$  is a power of two. To see the parallels to Section 4, think of  $m = n$ . For any  $a$ , we define  $\mu_a > 0$  to be the smallest value such that  $h_{a,0}^0(\mu_a) \leq 1/(2n)$ . Then  $h_{a,0}^0([n])$  falls in  $\mu_a$  equidistant intervals, each of length at most  $1/(2\mu_a)$ . This leaves us with  $\mu_a$  equidistant empty intervals, each of length at least  $\leq 1/(2\mu_a)$ . Together these empty intervals cover half the cyclic unit interval. When we add  $b$  it has the effect of placing 0 randomly on the cycle. Having chosen  $a$  and  $b$ , a random  $q$  is also hashed to random place on the cycle. The probability that  $q$  and 0 end in the same empty interval and with  $q$  after 0 in the interval is  $1/(2^3\mu_a)$ . In this case,  $q$  has the minimal hash value, but that should only have happened with probability  $1/n$ . Thus, relatively speaking, the probability is too high by a factor  $\Omega(n/\mu_a)$ , matching the linear probing cost from Section 4. As a result we will also end up concluding that the expected min-probability is too high by a factor  $\Omega(\lg n)$ .

As in Section 4, there are some details to consider. It is convenient to restrict ourselves to odd values of  $a$ ,  $b$ ,  $x = \mu_a$  and  $q$ . As a result, all our hash values are odd, and at odd multiples of  $1/2^\ell$  in the cyclic unit interval. The analysis goes through as long as  $\ell > \lceil \log_2 n \rceil$ , and in fact, we can shift out all but the  $\lceil \log_2 n \rceil$  most significant bits and yet have the same lower bound that the min-probability is too high by a factor  $\Omega(\lg n)$ .

## References

- [1] Noga Alon and Asaf Nussboim.  $k$ -wise independent random graphs. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 813–822, 2008.
- [2] John R. Black, Charles U. Martel, and Hongbin Qi. Graph and hashing algorithms for modern architectures: Design and performance. In *Proc. 2nd International Workshop on Algorithm Engineering (WAE)*, pages 37–48, 1998.
- [3] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000. See also STOC’98.
- [4] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29:1157–1166, 1997.

- [5] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997. See also STOC’94.
- [6] Martin Dietzfelbinger. Universal hashing and  $k$ -wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.
- [7] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.
- [8] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with  $O(1)$  worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. See also FOCS’82.
- [9] Gregory L. Heileman and Wenbin Luo. How caching affects hashing. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)*, page 141154, 2005.
- [10] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001. See also SODA’99.
- [11] Donald E. Knuth. Notes on open addressing. Unpublished memorandum. See <http://citeseer.ist.psu.edu/knuth63notes.html>, 1963.
- [12] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
- [13] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear probing with constant independence. *SIAM Journal on Computing*, 39(3):1107–1120, 2009. See also STOC’07.
- [14] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. See also ESA’01.
- [15] Mihai Pătraşcu and Mikkel Thorup. On the  $k$ -independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 715–726, 2010.
- [16] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC’11.
- [17] Jeanette P. Schmidt and Alan Siegel. The analysis of closed hashing under limited randomness. In *Proc. 22nd ACM Symposium on Theory of Computing (STOC)*, pages 224–234, 1990.
- [18] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. See also SODA’93.
- [19] Alan Siegel and Jeanette P. Schmidt. Closed hashing is computable and optimally randomizable with universal hash functions. Technical Report TR1995-687, Currant Institute, 1995.
- [20] Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proc. 11th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–497, 2000.

- [21] Mikkel Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. 45th ACM Symposium on Theory of Computing (STOC)*, 2013.
- [22] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012. Announced at SODA’04 and ALENEX’10.
- [23] Mark N. Wegman and Larry Carter. New classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. See also FOCS’79.